Juha-Matti Klapuri

Larox LSF -suodattimen ohjelma ja käyttöliittymä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

9.5.2016

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Juha-Matti Klapuri Larox LSF -suodattimen ohjelma ja käyttöliittymä 31 sivua 9.5.2016
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja(t)	lehtori Keijo Länsikunnas automaation päällikkö Timo Hietikko

Insinöörityössä oli tavoitteena rakentaa uusi ohjelma ja käyttöliittymä Outotecin LSF-suodattimelle, käyttäen uutta CODESYS-ohjelmistoalustaa. Pohjana käytettiin ennestään löytyviä vanhoja määrityksiä. Ohjelman pohjana käytettiin Siemensin S7-versiota, ja käyttöliittymän pohjana vanhaa Proface-versiota. CODESYS:iin rakennettiin molemmat osat.

Ohjelma kirjoitettiin IEC 61131-3 -standardiin kuuluvilla Ladder Logic- ja ST-kielillä, ja käyttöliittymä CODESYS:in omalla visualisaatioeditorilla. Laitteina käytettiin Janztecin teollisuus- ja paneeli-PC:itä, joissa pyörii Ubuntu Linux -käyttöjärjestelmä. Ohjelmien rakennetta verrattiin MVC-suunnittelumalliin ja miten se voisi toteutua tässä ympäristössä. PC:t yhdistettiin ethernetin kautta toisiinsa ja kommunikaatio toteutettiin CODESYS:in omalla DataServer-työkalulla, joka pyöri käyttöliittymän puolella.

Työn lopputuloksena saavutettiin toimiva ohjelma ja käyttöliittymä. Erityisesti käyttöliittymä näytti paremmalta ja tarkemmalta verrattuna vanhaan versioon. Pari ongelmaa kuitenkin jäi lopuksi. Käyttöliittymän banner-ominaisuus ei toiminut toivotulla tavalla, ja hälytysten historia-toiminto ei toiminut. Nämä korjattiin jälkikäteen. MVC-suunnittelumalli toteutui kohtalaisesti. Syntynyttä ohjelmaa ja käyttöliittymää voidaan käyttää jatkossakin muissa suodattimissa, kunhan käytetty CODESYS-ohjelmisto päivitetään uudempaan.

Avainsanat	automaatio, MVC, CODESYS, IEC 61131-3, Linux

Author(s) Title Number of Pages Date	Juha-Matti Klapuri Larox LSF –filters program and user interface 31 pages 9 May 2016
Degree	Bachelor of Engineering
Degree Programme	Computer Science
Specialisation option	Software Engineering
Instructor(s)	Keijo Länsikunnas, Senior Lecturer Timo Hietikko, Chief of Automation

The objective of this thesis was to build a new program and user interface for Outotec LSF-filter by using the new CODESYS-platform. Existing specifications were used as the base. The old Siemens S7 – version was used as a base for the program, and Proface-version as a base for the interface. Both parts were created in CODESYS.

The program was written in Ladder Logic and ST-languages which belong to the IEC 61131-3 -standard, and the interface was made with CODESYS Visualization Editor. Janztec industrial- and panel-PC:s which run Ubuntu Linux -operating system were chosen for the hardware. The programs structure was compared to MVC-design model and how it would be implemented in the environment. The computers were connected with Ethernet and communication was made with CODESYS DataServer-tool, which runs on the interface side.

The thesis resulted in a working program and interface. Especially the interface looked better and sharper compared to the old version. A few problems were left however, the banner function in the interface didn't work as intended and the alarm history didn't work. These were fixed afterwards. The MVC-model was implemented moderately. The created program and interface can be used in other filters as long as the CODESYS-platform is updated to a newer version.

Keywords	automation, MVC, CODESYS, IEC 61131-3, Linux

Sisällys

Lyhenteet

PLC	Programmable logic controller. Ohjelmoitava logiikka.
FBD	Function Block Diagram. Yksi käytettävistä ohjelmointikielistä.
LD	Ladder Logic. Yksi käytettävistä ohjelmointikielistä.
ST	Structure Text. Yksi käytettävistä ohjelmointikielistä.
IL	Instruction List. Yksi käytettävistä ohjelmointikielistä.
SFC	Sequential Function Chart. Yksi käytettävistä ohjelmointikielistä.
POU	Program Organisation Unit, ohjelman organisaatioyksikkö
PROG	Program, ohjelma
FB	Function Block, funktioblokki
FUNC	Function, funktio

1 Johdanto

Työn aiheena on Outotecin 'Larox LSF' -suodattimen ohjelman ja käyttöliittymän päivittäminen uudelle alustalle. Tavoitteena on luoda toimiva ohjauspuoli, joka pyörii sulavasti annetun päivityssyklin puitteissa, sekä luoda selkeä käyttöliittymä. Ohjelmistona ja alustana toimii CODESYS, joka korvaa vanhan Siemensillä tehdyn ohjelman näihin suodattimiin.

Suodatin itsessään on ontto sylinteri, joka sisältää kangaspusseja. Näihin pumpataan lietettä, ja adsorption avulla neste valutetaan pois ja haluttu kiinteä aines jää pussiin talteen.

Tähän työhön kuuluu yhteensä kolme suodatinta, joille tehdään ensiksi yksi yhteinen ohjelma, joka muokataan lopuksi erikseen sopivaksi jokaiselle. Suodattimen hallintaohjelma ohjelmoidaan yhdistelmällä Ladder logic -kieltä ja Pascalia muistuttavaa ST-kieltä. Lopullista ohjelmaa voidaan käyttää myös myöhemmissä suodattimissa. Käyttöliittymä tehdään myös CODESYSillä, sen omalla käyttöliittymäeditorilla, jolle ohjelmoidaan toiminnallisuuksia edellämainituilla kielillä. Ohjelma ja käyttöliittymä asennetaan erikseen kahdelle koneelle, jossa alustana toimii Linux. Yhteys koneiden välillä tulee toimimaan ethernetin välityksellä.

Työn tilaaja on Etteplan Design Center Oy, mikä suunnittelee teollisia laitteistoja, teknisen tuoteinformaation ratkaisuja ja palveluja. Työ on yksi monista Etteplanin automaation projekteista, ja koska kirjoittajalla on jo työharjoittelun kautta kokemusta vastaavista töistä, oli aiheen valinta insinöörityöksi helppo.

2 IEC 61311 -standardi

2.1 Standardin aihepiiri

Ennen kuin insinöörityön kulkua voidaan selostaa, on hyvä ensin tietää, mistä ohjelmiston perustat tulevat. IEC (International Electrotechnical Commission) 61311 on

kansainvälinen standardi, joka määrittelee ohjelmoitavien logiikoiden laitteistojen vaatimukset, ohjelmointikielet ja muut vaadittavat kehykset. Nämä kehykset ovat löyhiä ja antavat eri valmistajille paljon pelivaraa tehdä omia toteutuksiaan. Esimerkiksi Siemensin Simatic-ohjelmistossa on erilainen toteutus aikafunktioille ja CODESYSillä omansa.

2.2 IEC 61311-3- ja ohjelmointikielet

Tässä työssä oleellinen osa-alue on IEC 61131-3, joka määrittelee työssä käytettävät ohjelmointikielet. Eri ohjelmointikieliä, joita voi käyttää, on peräti viisi kappaletta:

- FBD (Function Block Diagram)
- LD (Ladder Diagram)
- ST (Structure Text)
- IL (Instruction List)
- SFC (Sequential Function Chart).

Näistä FBD, LD ja SFC ovat graafisia kieliä, IL ja ST ovat puhdasta kirjoitettua tavallista koodia. IL muistuttaa paljon alemman tason konekieltä, kun taas ST on korkean tason ohjelmointikieli, joka muistuttaa Pascalia. LD ja FBD eroavat toisistaan lähinnä graafisesti. LD muistuttaa vanhoja relekaavioita, joita käytettiin ennen tietokoneita automaation toteutuksissa. FBD esittää loogiset ja aritmeettiset operaatiot eri tavalla verrattuna LD:iin. Useimmissa tapauksissa LD- ja FBD- näkymät voidaan vaihtaa keskenään rikkomatta koodia. Lopuksi on SFC, joka kuvaa ohjelman asteittaista ja samanaikaista toteutusta.

Insinöörityössä tullaan käyttämään näistä lähinnä kahta: Ladder Diagramia ja Structure Textia. LD:tä käytetään siksi, koska vanha Siemensin versio on tehty sillä, ja se on myös tutumpi muille näissä projekteissa työskenteleville. ST:llä kirjoitetaan funktioita ja funktioblokkeja, joita voidaan kutsua LD:llä tehdyissä ohjelmissa.

2.3 Program Organisation Unit

Program Organisation Unit, ohjelman organisaatioyksikkö eli POU, on standardin määrittelemä koodilohko. Nämä lohkot muodostavat ohjelman koko koodikokonaisuuden, joilla suodatinkin tulee pyörimään. Lohkoja on kolmea erilaista: Function (FUN, funktio), Function Block (FB, funktiolohko) ja Program (PROG, ohjelma).

2.3.1 Function Block

Function Block on ehkäpä tärkein POU, edellä mainituista kolmesta. CODESYS ja muut vastaavat ohjelmistot tarjoavat vakiona tietyt perus-FB:t, esimerkiksi kaikki tavalliset aritmeettiset operaatiot kuten yhteenlasku, kertolasku ja niin edelleen. Ohjelmalla voi tehdä myös omia funktioblokkeja, jolle voi ohjelmoida omat operaationsa, input- ja output-muuttujansa ja niin edelleen. Funktioblokkeja voi siis olla useita instansseja.

2.3.2 Function

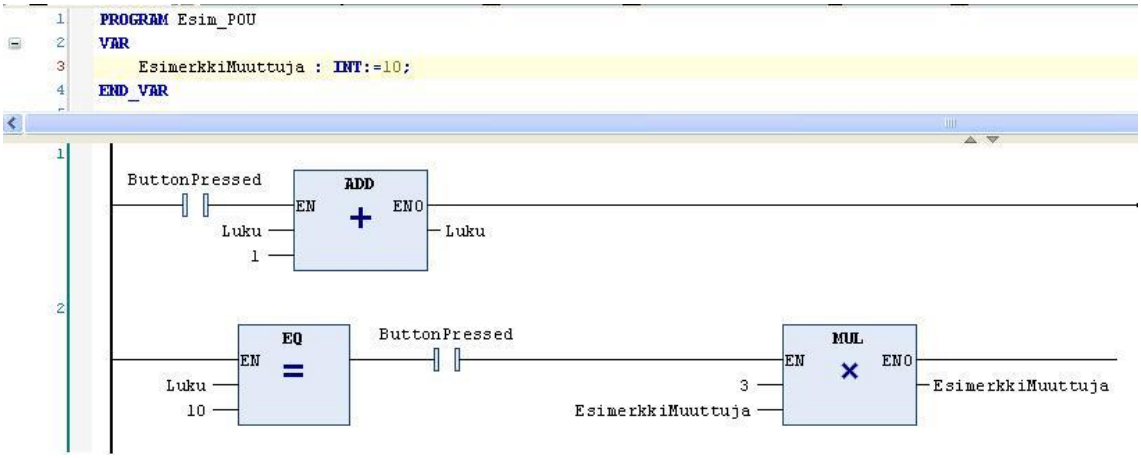
Function eli funktio eroaa Function Blockista lähinnä muuttujien osalta. Funktiolla ei ole pysyviä muuttujia, jotka pysyisivät ohjelman muistissa. Funktiolle voidaan välittää parametrit, joiden avulla se toteuttaa sille kirjoitetun koodinsa. Lopuksi funktio palauttaa jonkin return-arvon, joka voidaan välittää ohjelmassa eteenpäin.

2.3.3 Program

Program eli ohjelma on kolmas listatuista POU:sta, ja hierarkialtaan ylin. Ohjelma koostuu funktioblokeista ja funktioista, ja mahdollisesti muista aliohjelmista, jotka voivat myös kutsua muita POU:ita. CODESYSissä määritellään asetuksista, mikä ohjelma aloittaa suorituksen, eli niin sanotun Taskin eli tehtävän.

2.4 Esimerkkikoodia

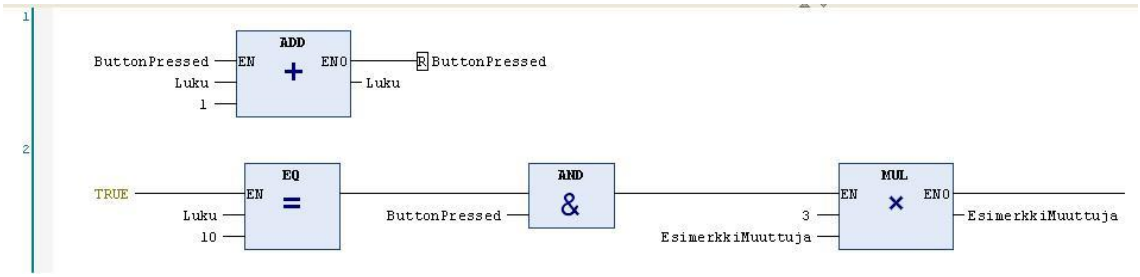
Edellä selostettujen asioiden avulla voidaan tehdä ja ymmärrettävästi esittää toimivaa koodia. Seuraavaksi esimerkki koodista, jossa suoritetaan nappia painamalla yhteen- ja kertolaskuja.



Kuva 1. Ladder Diagramilla tehtyä koodia

Kuvassa 1 on kirjoitettu Ladder Diagramilla pieni pätkä koodia. Yläosassa on esitelty ensin ohjelma nimeltä Esim_POU, jonka jälkeen listaus paikallisista muuttujista, joita on yksi kappale. EsimerkkiMuuttuja on tyyppiä INT, joka on alustettu arvolla 10. Tätä muuttujaa ei voida nähdä Esim_POU:n ulkopuolella.
Seuraavaksi nähdään graafista Ladder Diagram -koodia. Kuvassa on kaksi Networkia, eli verkkoa, numeroituna 1 ja 2. Verkossa 1 on yksi ”rele” ja yksi ADD-operaatio, joka on siis tavallinen funktioblokki. Rele toimii ns. if-lauseena, ja sille on annettu muuttujaksi ButtonPressed. Eli kun ButtonPressed on TRUE, ohjelma jatkaa siitä eteenpäin ja toteuttaa yhteenlaskuoperaation, jossa muuttujaa Luku kasvatetaan yhdellä. Lopuksi ohjelma jatkaa eteenpäin ja alustaa muuttujan ButtonPressed takaisin nollaan (ei näy kuvassa).

2. verkossa on hieman monimutkaisempi rakenne: tällä kertaa on kaksi ehtoa, jotta ohjelma toteuttaa MUL-operaation eli kertolaskun. Aluksi muuttujan Luku on oltava täsmälleen 10, ja muuttujan ButtonPressed on oltava TRUE. Kun nämä ehdot toteutuvat, EsimerkkiMuuttuja kerrotaan kolmella ja talletetaan samaan muuttujaan.



Kuva 2. Sama koodi kuin yllä, mutta FBD-näkymässä

Kuvassa 2 on täsmälleen sama koodi kuin kuvassa 1 mutta tällä kertaa esitettynä Function Block Diagramilla. Erot ovat oikeastaan vain graafisia. Kuvasta näkee myös 1. verkon reset-toiminnon ButtonPressed-muuttujalle. 2. verkossa havainnoituu ehkä paremmin, että kertolaskuun vaaditaan kaksi ehtoa, joiden pitää toteutua.

```
1  IF ButtonPressed = TRUE THEN
2      Luku := Luku + 1;
3      ButtonPressed := FALSE;
4  END_IF
5
6  IF ButtonPressed = TRUE AND Luku = 10 THEN
7      EsimerkkiMuuttuja := EsimerkkiMuuttuja * 3;
8  END_IF
```

Kuva 3. Sama koodi ST:llä kirjoitettuna

Viimeiseksi vielä on sama logiikka kirjoitettuna ST:llä. Jos ohjelmalohkossa ei ole monimutkaisia rakenteita tai toimintoja, niin kuvan kaltaiset yksinkertaiset operaatiot voivat olla näppärämpiä kirjoittaa ST:llä.

3 CODESYS ja suodatinjärjestelmän hierarkia

3.1 Miksi uusi alusta

Syy uuden CODESYS-version tekeminen ohjelmasta on varsin yksinkertainen. Vanhaa Siemens-mallia ei yksinkertaisesti voida toimittaa loppuasiakkaalle, joka synnytti tarpeen vaihtoehtoiselle alustalle. Asiakas on tutkinut eri alustavaihtoehtoja, ja lopulta päätynyt CODESYSiin. Päätökseen on vaikuttanut sopivan laitteiston saatavuus ja sopivuus alustalle, sekä tietysti hinta.

Alustan vaihtoa voitaisiin pitää jopa hieman erikoisena. Teollisuudessa ei kovin nopeasti haluta siirtyä uudempaan ja tehokkaampaan tekniikkaan. Esimerkiksi Siemensin ja muiden valmistajien teollisuus-PC:ssä ja laitteissa saattaa pyöriä hyvinkin vanhaa tekniikkaa, kuten 90-luvun prosessoreita, jopa alikellotettuna. Luotettavuus on tärkein asia näissä tapauksissa, tehokkuuden edellä. Jos Siemensin tuttuja ja luotettavia versioita olisi voitu toimittaa, tälle työlle ei varmaan olisi ollut tarvetta, koska ohjelmat olisivat olleet valmiita ennestään.

3.2 Mikä on CODESYS?

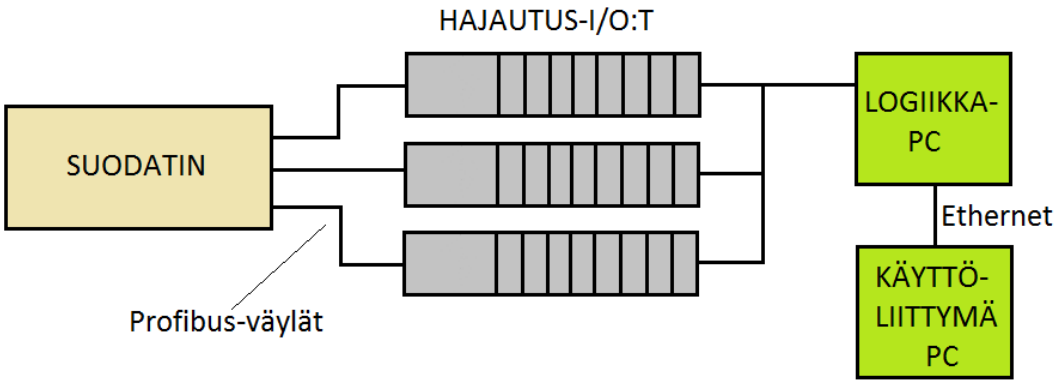
CODESYS on saksalaisen 3S-Smart Software Solutionsin kehittämä ohjelmistoympäristö erityisesti teollisuuden automaation projekteihin. Yhtiö on perustettu vuonna 1994 ja ensimmäinen versio CODESYSistä on julkaistu vuonna 1996. Lisenssejä on myyty tähän asti yli kolme miljoonaa ympäri maailman ja teollisuuden asiakkaita on noin 350. Ohjelman uusin pääversio V3 julkaistiin vuonna 2006, mitä käytetään tässä työssä.

CODESYS tarjoaa kaiken tarvittavan, mitä tarvitaan ohjelman tekemiseen ja sen ajamiseen. Ohjelma kirjoitetaan CODESYSin kehitysympäristössä. Lisäksi käyttöliittymän suunnittelu tapahtuu samalla työkalulla. Toisilla ohjelmistoilla käyttöliittymä tehdään usein jollain täysin erillisellä ohjelmalla. Esimerkkinä tästä on, että suodattimen vanha ohjelma on tehty Siemensin Simaticilla, ja käyttöliittymä taas Proface-nimisellä ohjelmalla.

Kun ohjelma ja käyttöliittymä saadaan valmiiksi, ne on ladattava ajoympäristöön (runtime). CODESYS tarjoaa tähän myös oman ratkaisunsa ja sitä käytetään suodattimessa. Ohjelman toimivuus vaatii lisäksi Outotecin omat ajurit, jotta ajoympäristö ymmärtää kommunikoida oikein fyysisen laitteen kanssa.

3.3 Suodatinjärjestelmän hierarkia

Koko järjestelmä voidaan jakaa seuraavanlaisesti osiin: suodattimeen, hajautus-I/O:ihin, tietokoneisiin, joista toisessa CODESYSillä on tehty ohjauslogiikka ja toisessa käyttöliittymä.



Kuva 4. Suodatinjärjestelmän hierarkia kokonaisuudessaan

Suodatin on itsessään vain ontto pönttö, jossa on sisällä kangaspusseja. Sen toimintaa ohjataan pöntön ulkopuolelta venttiileiden, moottoreiden ja sensoreiden avulla. Laitteissa on tietty määrä sisääntuloja ja lähtöjä, joista saadaan dataa. Nämä kytketään hajautus-I/O:ihin Profibus-kenttäväylän kautta.

Kuva 5. Suodatin avattuna pönttö näkyy oikealla ja suodatinpussit vasemmalla

Hajautus-I/O koostuu karkeasti väyläohjaimesta, digitaalisista sisääntuloista ja lähdöistä sekä analogisista sisääntuloista ja lähdöistä. I/O-laite on teline, tai räkki, joka koostuu tietystä määrästä kortteja, joita voidaan asentaa siihen. Yhdessä kortissa on yleensä yhdestä kahdeksaan kappaletta digitaalisia tai analogisia sisääntuloja tai lähtöjä.

Kuva 6. Wagon valmistama hajautus-I/O

Viimeisenä osana hierarkian päässä on tietokone, jossa pyörii Linux-käyttöjärjestelmä. Tietokone on erityinen teollisuudelle suunniteltu malli, joka asennetaan kaappiin. Tähän koneeseen asennetaan CODESYS-ajoympäristö, joka pyörittää ohjelmalogiikkaa sekä käyttöliittymää. Ohjelmalogiikka saa I/O-järjestelmältä tietoa I/O:iden tilasta, ja käyttäytyy niiden mukaan. Käyttöliittymästä suodatinta voidaan säätää ja kontrolloida.

4 Suodattimen ohjelma

4.1 Määrittely

Insinöörityön käytännön puoli koostuu laajalti suodattimen ohjauslogiikan ja käyttöliittymän tekemisestä, ja näiden kytkemisestä toisiinsa. Tekijän omien kokemuksien perusteella käyttöliittymä kannattaa tehdä erillään ohjelmapuolesta, jotta sitä voidaan testata tehokkaammin ja helpommin. Koska projekti sisältää nimenomaan nämä osapuolet, voitaisiin soveltaa ohjelmistokehityksessä yleisesti käytettyä MVC-suunnittelumallia ohjelman teossa. MVC-suunnittelumallissa ohjelma jaetaan kolmeen osaan, jotka ovat

- malli (model)

- näkymä (view)

- käsittelijä (controller).

Ensimmäinen osa, malli, sisältää ydintoiminnallisuuden eli ohjelman logiikan, tiedot ja säännöt. Tavallisessa tietokoneohjelmassa malli sisältäisi tarvittavat funktiot, joilla näkymät saisivat vastaanotettua tarvittavan tiedon mallilta sekä proseduurit, joita käsittelijät kutsuvat.

Näkymä näyttää tietoa käyttäjälle, joka vastaanotetaan mallilta. Aina kun mallissa jokin näkymän tarvitsema tieto muuttuu, se välitetään sille päivitettynä. Näkymä sisältää myös käsittelijän, joka toimii taustalla.

Viimeinen osa, käsittelijä, nimensä mukaisesti käsittelee käyttäjän antamat komennot. Komennot voivat olla esim. hiiren painalluksia. Käsittelijä välittää nämä joko mallille, jossa tehdään haluttu muutos, tai komento voi olla vain esim. sivunvaihdos, jolloin mallille ei tarvitse välittää pyyntöä.

CODESYS-ohjelmassa MVC-malli toteutuisi seuraavanlaisesti. Ohjelmapuoli ts. logiikka toimisi mallina. Se pyörittää koko suodatinta automaattisesti riippumatta käsittelijöistä ja näkymistä. Näkyminä toimivat CODESYS:in visualisaatiosivut, joista jokainen on yksi näkymä. Visualisaatiosivut vastaanottavat DataServerin kautta logiikalta päivitetyt tiedot muuttujien tilasta. Lopuksi käsittelijät toteutetaan visualisaatoissa, elementteihin kuten nappeihin voidaan sisällyttää koodia. Esimerkiksi Wash-nappia painamalla asetetaan muuttuja, joka välitetään taas DataServerin (tästä myöhemmin) kautta logiikalle, joka lopulta suorittaa koodilohkon joka käynnistää pesun.

Projekti myös ”lukitaan” yhteen CODESYS-versioon. Edellisten kokemusten perusteella päivittäminen on tuonut outoja ongelmia vanhemmalla versiolla tehtyyn projektiin. Luonnollisesti jos kriittisiä päivityksiä tulee, jotka korjaavat jotain tärkeää niin silloin ohjelmisto päivitetään. Mahdollisen päivityksen jälkeen on varmistettava, ettei mikään projektin osa ole päivityksen takia hajonnut. Projekti on siis testattava, ja aiemmin ilmenneiden ongelmien testausmenetelmänä voidaan käyttää regressiotestausta. Regressiotestauksessa käytetään edellisten versioiden testausskenaarioita, joilla vanhat ongelmat ovat aikaisemmin ilmenneet. Ideana on siis varmistaa, että vanhoja bugeja ei

ole palautunut (tai uusia ilmennyt) päivityksen jälkeen. Tällä tavalla paljastuneita bugeja kutsutaan tällöin regressioiksi.

Ohjelmalogiikan tekoa ei lähdetä tekemään aivan tyhjältä pohjalta. Koko projektin tarkoituksena on tehdä suodattimen ohjelmasta ja käyttöliittymästä versio, joka pyörii uudella CODESYS-alustalla. Ohjelma tehdään olemassa olevien vanhojen määritysten mukaisesti, jotka ovat samat alustasta riippumatta.

4.2 Suunnittelu

Ohjelmalogiikan tekoa lähdettiin suunnittelemaan ensin tutkimalla vanhaa Simaticin koodia. Käytännössä koodi halutaan mahdollisimman samankaltaiseksi uuteen versioon, Koodin lisäksi myös muuttujat tuodaan Simaticista CODESYSiin. Logiikkakoodi on Simaticissa sijoiteltu sen omiin funktioihin, ja muuttujat on sijoiteltu tietokantoihin (DB, database).

CODESYSissä logiikkakoodi kirjoitetaan Programeihin, joita voi olla useampi. Melkein kaikki muuttujat tallennetaan Global Variable Listeihin (GVL, globaalimuuttujalista), ja loput Structeihin (STRUCT, struktuuri) kuten esimerkiksi hälytysten muuttujat. Structit mahdollistavat muuttujien käsittelemisen eri tavalla tietyissä tapauksissa.

4.3 Toteutus

4.3.1 Muuttujat

Ensin lähdin miettimään, miten saisin muuttujat järkevimmin tuotua Simaticista CODESYSiin. Yksinkertainen kopio-liitäntä ei ole mahdollista, koska muuttujien syntaksi on erilainen Simaticissa. Lisäksi muuttujia on varmaan ainakin noin 1500 kappaletta, joten niiden käsin kirjoittaminen uuteen ohjelmaan aikarajojen puitteissa veisi aivan liikaa aikaa.

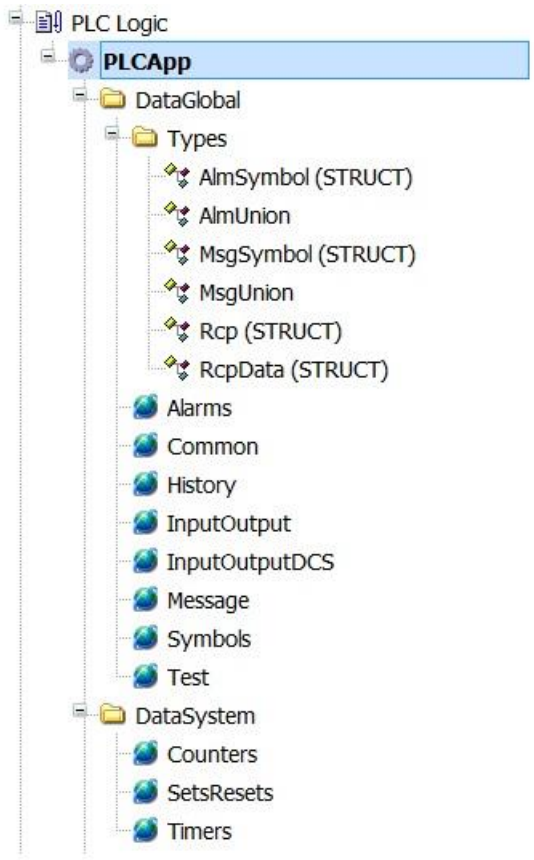
Selvitetään, miksi muuttujia on niin paljon ja mistä osa tulee. Ensiksi meillä on fyysiset I/O-kortit, joista tulee ja joihin lähetetään muuttujien tietoja, kutsutaan näitä hardware-muuttujiksi. Digitaalisia I/O-kortteja on 21 kappaletta, joissa jokaisessa on 8 lähtöä tai tuloa. Pelkästään näistä saadaan jo 168 muuttujaa ohjelmaan. Seuraavaksi on

analogiset kortit, joita on viisi kappaletta. Näistä jokaisessa on kaksi kanavaa. Eli analogisia tietoja tulee 10, mutta määrä tuplaantuu, koska ohjelmaan on luotava jokaiselle erillinen muuttuja, johon talletetaan tieto fysikaalisena yksikkönä (metri, paine jne.). Analogitieto tulee tyypillisesti 16-bittisenä sanana, jonka arvorajat liikkuvat -32768:sta 32768:een, tai 0:sta 65536:een, jos etumerkkiä ei käytetä. Tätä kutsutaan raaka-arvoksi, joka sitten muunnetaan ohjelmassa laskukaavalla fysikaaliseksi yksiköksi kuten baareiksi tai celsius-asteiksi.

I/O-korteilta tulee siis noin 180 muuttujaa ohjelmaan, ja tämä määrä pitää vielä kolminkertaistaa. Ensiksi hardware-muuttujat kopioidaan erillisiin muuttujiin, joita varsinaisesti käsitellään logiikassa, esimerkiksi hardware-muuttuja X11DI03CH01 kopioidaan Symbols.M42R-muuttujaan, joka on moottorin M42 käymistieto. Näin estetään koskeminen suoraan korttien muuttujiin sekä käytetään muuttujia, joita on helpompi ja selkeämpi käyttää itse ohjelmassa. Lisäksi tällä metodilla on helppo suorittaa simulointia, kun fyysiset hardware-muuttujat voidaan yksinkertaisesti jättää siirtämättä koodissa, jolloin vain suoritetaan puhtaasti ohjelmallista koodia. Hardware-muuttujat kopioidaan vielä kolmannen kerran, koska kaikki hardware-tiedot pitää siirtää myös DCS (Distributed Control System) -järjestelmään. Tämä on asiakkaan oma järjestelmä, jonne yhteys tehdään Modbus-protokollalla. Tarkemmin ModbusTCP:llä, joka välitetään logiikkakoneen toisen ethernet-väylän kautta. DCS-muuttujiin yksinkertaisesti kopioidaan ohjelmassa I/O-korttien muuttujatiedot.

Hardware-muuttujien lisäksi ohjelmassa käytetään tietysti myös tavallisia ohjelmallisia muuttujia, joita ei ole kytketty mihinkään fyysiseen laitteeseen. Tähän joukkoon kuuluu esim. käyttöliittymästä tai DCS-järjestelmästä tulevat käskybitit, analogitiedoista talteen otettuja arvoja kalibrointia varten, ajastimia ja niin edelleen. Näiden muuttujien lukumäärä kohoaa noin 800:aan ja kokonaismäärä kohoaa noin 1300:aan.

Ratkaisu muuttujien siirtämiseen Simaticin tietokannasta on tehdä Excel-taulukot, joissa tehdään tarvittavat muutokset. Muuttujat tuodaan Simaticista Exceliin sellaisenaan. Excelin funktioiden avulla muuttujat, jotka ovat nyt käytännössä Stringejä Excelissä, pilkotaan osiin. Näistä osista rakennetaan siten taas Excelin funktioiden avulla uusia Stringejä, joiden syntaksi vastaa CODESYSin koodia. Uudet listat voidaan sitten viedä CODESYSiin ilman virheitä.



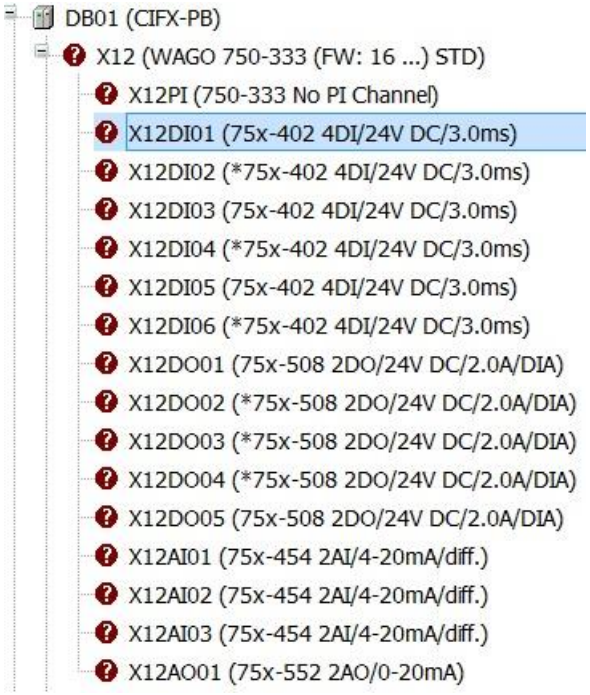
Kuva 7. Muuttujalistat.

Kuvassa 7 näkyvät ohjelman muuttujien listat. Symbols-lista sisältää lähes kaikki Boolean-tyyppiset muuttujat ja lista on muuten tavallista VAR_GLOBAL-tyyppiä. Common- ja History-listat on sen sijaan määritelty VAR_GLOBAL RETAIN-tyyppiseksi. Tämä tarkoittaa, että jos suodatin tai ohjelma sammuu jostain syystä, uudelleenkäynnistyksessä muuttujat muistavat arvonsa, mitkä niillä oli sammutuksen yhteydessä. Nämä listat sisältävät siis lähinnä muita muuttujia kuin Boolean-tyyppejä, kuten integerejä. History-lista sisältää lähinnä aikatietoja kuten miten pitkään suodatin on ollut päällä jne.

Vielä yksi muuttujatyyppi on VAR_GLOBAL PERSISTENT RETAIN-tyyppiset muuttujat. Nämä sijoitetaan aivan erikseen omaan Persistent Variables -listaan. Nämä eroavat RETAIN-tyyppisistä muuttujista siten, että niiden arvot eivät resetoidu koskaan, ainoastaan jos koko ohjelma halutaan nollata. RETAIN- ja RETAIN PERSISTENT -muuttujille määritellään muistissa oma alue.

Toinen tärkeä asia on kuvassa näkyvät Structit. Hälytysten muuttujat on esitelty AlmSymbol-structissa eikä tavallisessa muuttujalistassa. Seuraavaksi on AlmUnion, joka on Union-tyyppinen datatyyppilista. Union toimii siten, että siinä on esitelty edellä mainittu AlmSymbol-structi sekä Array, jossa on 128 Boolean-tyyppistä paikkaa. Käytännössä tämä toimii niin, että jos ohjelma muuntaa esimerkiksi Arrayn ensimmäistä alkiota, muutos näkyy myös AlmSymbolissa. Tällä on siis se hyöty, että hälytykset voidaan käydä läpi Arrayn avulla for-loopissa. Lopuksi Alarms-listassa esitellään AlmUnion, ja tätä käytetään ohjelman kutsuissa.

Kuvassa näkyy myös DataSystem-hakemisto, joka sisältää laskureita ja ajastimia. Näiden tietotyypit ovat CODESYSin omia, eikä niitä ole voitu tuotu Simaticista.



Kuva 8. Yksi projektiin määritelty hajautus-I/O

Kuvassa näkyy yksi suodattimen hajautus-I/O, joka sisältää osan laitteen tuloista ja lähdöistä. DI- ja DO-päätteiset ovat digitaalisia tuloja ja lähtöjä, AI- ja AO-päätteiset analogisia. Jokaiselle tulolle sekä lähdölle on määritelty ohjelmassa yksi muuttuja, joka saa niiden arvon. Nämä muuttujat on esitelty InputOutput- sekä InputOutputDCS-muuttujalistoissa. Lopuksi koodissa muuttujat kytketään esimerkiksi Symbols-listassa esiteltyihin muuttujiin.

4.3.2 Reseptit

Suodatinta ajetaan tietyllä valitulla ohjelmalla, ja näitä ohjelmia kutsutaan resepteiksi. Reseptejä voidaan ajatella samalla tavalla, kuten esimerkiksi pyykinpesukoneesta löytyviä eri pesuohjelmia. Työn suodattimessa resepti koostuu aliohjelmista (subroutines), joista joku on aina ajossa, kun suodatin on päällä. Aliohjelmia ovat esimerkiksi suodatus ”tavalla 1” ja pesu.

Ohjelmallisesti resepti koostuu yksinkertaisesti structista, jossa on yksi taulukko kokonaislukuja ja kaksi taulukkoa reaalilukuja. Reaalilukuihin voitaisiin tallettaa lisäparametrejä tarvittaessa, mutta oleellisin on kokonaislukutaulukko, johon talletetaan aliohjelmien numerot eli mitä niistä halutaan kyseisellä reseptillä ajaa.

Reseptikokonaisuudet talletetaan vielä yhteen structiin, joka säilöö yhteensä 10 kappaletta reseptejä. Reseptejä voidaan sitten valita suoraan käyttöliittymästä reseptisivulta. Työn suodattimessa ennalta määriteltyjä reseptejä ei ole kuin pari kappaletta, mutta muissa niitä voisi olla enemmän, jolloin on perusteltua ylläpitää rakenteita, joihin mahtuu enemmän kuin tarvitaan.

CODESYS-kehitysympäristössä resepteillä on myös oma merkityksensä. Tällöin resepteillä tarkoitetaan valmiiksi talletettuja arvoja eri muuttujille. CODESYS tarjoaa tähän Recipe Manager -työkalun eli reseptimanagerin. Reseptimanageri yksinkertaisimmillaan tallettaa muuttujan sekä sen arvon, mutta tarjoaa myös esimerkiksi minimi- ja maksimiarvon talletuksen monimutkaisempiin ratkaisuihin. Oletusratkaisu riittää tähän projektiin. Reseptimanageri voi säilöä tietonsa joko tekstitiedostona tai binäärinä.

Reseptimanageri tässä projektissa on lähinnä tukemassa muuttujien säilytystä. Alun perin manageria ei käytetty, vaan kaikki pysyvässä muistissa pidettävät muuttujat säilytettiin vain PersistentVars-muuttujalistassa, PERSISTENT RETAIN -tyyppisenä. Testeissä huomattiin kuitenkin, että tietokoneen uudelleenkäynnistyksen jälkeen ajossa tehdyt muutokset parametreihin eivät säilyneet, vaan ne palautuivat alkuperäisiin arvoihinsa. Reseptimanagerin käyttöönotto korjasi tämän ongelman.

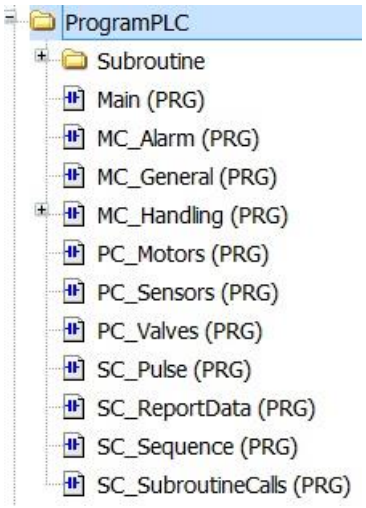
4.3.3 Logiikka

Logiikkakoodi tuodaan vanhasta Simaticista CODESYSiin pohjaksi uudelle toteutukselle. Toisin kuin muuttujien kanssa, koodia ei myöskään voida vain kopio-liittää ohjelmasta toiseen, mutta sitä ei myöskään voida muokata Excelin kautta. Ainoa vaihtoehto on siis kirjoittaa logiikkakoodi itse käyttäen vanhaa koodia mallina.

Simaticissa koodi on sijoitettuna eri funktioihin, ja CODESYSissä koodit kirjoitetaan Programeihin. Näillä ei sen lisäksi ole eroja, joten kirjoittaminen onnistuu helpohkosti miettimättä erikoisia hierarkioita. Koodi jaetaan teemoittain eri Programeihin, kuten hälytykset MC_Alarm-ohjelmaan ja sensoreita koskevat asiat PC_Sensors-ohjelmaan.

Koodia lähdettiin kirjoittamaan järjestelmällisesti. Simaticin koodi mallina, kirjoitettiin koodia verkko verkolta käyttäen CODESYSin työkaluja. Tietyissä paikoissa tuli vastaan koodia, jota ei voitu kirjoittaa sellaisenaan uuteen ohjelmaan. Tällöin paikalle jätettiin kommentti, missä selitettiin, mitä ei voitu kirjoittaa.

Yksi iso eroavaisuus Simaticissa ja CODESYSissä on ajastimet ja laskurit. Pienellä opiskelulla ymmärrettiin niiden erot ja saatiin tehtyä korvaavat versiot uuteen logiikkakoodiin. Nämä on esitelty luvussa 4.3.1 mainitussa DataSystem-kansiossa.



Kuva 9. Logiikkakoodi sijoitettuna Programeihin.

Lopuksi tehtiin Main-ohjelma, joka kutsuu jokaista kuvan muuta ohjelmalohkoa, jolloin suodattimen kaikki osa-alueet pyörivät. Main-ohjelma määritellään Task Configurationissa pääohjelmaksi, jota kutsutaan käynnistyksessä. Lisäksi Taskille

määritellään 20 millisekunnin päivitysnopeus, eli koko ohjelma käydään aina sen ajan välein kokonaan läpi.

5 Käyttöliittymä

5.1 Määrittely

Käyttöliittymä rakennetaan alusta alkaen uusiksi CODESYSin ulkoasueditorilla, joka on osa muuta ohjelmistoympäristöä. Vanha ulkoasu oli tehty Proface-ohjelmistolla, joka ei edes olisi yhteensopiva CODESYSillä tehdyn ohjelmalogiikan kanssa. Asiakkaan toive käyttöliittymästä on yksinkertainen: sen on näytettävä samalta kuin vanha käyttöliittymä. Joitain eroavaisuuksia tulee kuitenkin olemaan, koska ohjelmat ovat varsin erilaisia, ja ne tekevät tietyt asiat eri tavalla, kuten esimerkiksi hälytysten esittämisen.

5.2 Suunnittelu

Ensimmäisenä käyttöliittymän suunnittelussa lähdetään sen resoluution määrittelyssä. Tämän määrää pitkälti tietokoneen näytön koko, joka on tämän projektin koneessa 1024x768. Vanhan Profacen versiossa on käytetty resoluutiota 800x600, joten uudesta tulee hieman tarkemman ja paremman näköinen. Ongelmana olisi voinut olla eri kuvasuhde, joka onneksi on sama näillä resoluutiolla. Ruudulle on täten helpompi sijoitella elementtejä oikeassa suhteessa.

Käyttöliittymä koostuu monesta eri sivusta, joiden välillä voi liikkua. Ne voidaan jakaa eräänlaisiin osiin seuraavasti: pääsivuihin, ohjesivuihin, säätö- ja testaussivuihin sekä kuvaajasivuihin.

Pääsivuilla näkee suodattimen sen hetkisen tilan, missä tilassa esimerkiksi venttiilit ovat, painemittareiden lukemat ja niin edelleen. Tilojen lisäksi pääsivuilta voi valita reseptin jota halutaan käyttää ja onko suodatin manuaalisessa vai automaattisessa tilassa. Lopuksi on sivu, joka näyttää hälytykset ja niiden tilat.

Ohjesivut sisältävät pitkälti suodattimen käyttöohjeet. Jokaiselle pääsivulle on oma ohjesivunsa, joka selostaa pääsivun sisällön toimintaa ja tarkoitusta. Lisäksi erikseen

löytyy manuaalisivu, jota voidaan selata, sekä ohjesivut hälytyksille. Hälytysohjeissa kerrotaan graafisesti, missä kohtaa suodatinta hälytys tapahtuu, miksi se tapahtuu ja toimenpiteet hälytyksen sattuessa.

Säätö- ja testaussivuilla voidaan säätää tiettyjä asioita suodattimesta, sekä testata suodattimen osia testausmoodissa. Testaussivut kattavat venttiileiden ja pumppujen käynnistämisen ja sammuttamisen sekä listan sisääntuloista ja lähdöistä tiloineen testimuodossa. Järjestelmä saadaan myös sammutettua näiltä sivuilta.

Viimeisenä osana ovat kuvaajasivut, jotka piirtävät paineen vaihteluita. Painetta mitataan kahdessa kuvaajassa. Toinen näyttää yhden tunnin paineenmuutokset ja toinen 24 tunnin ajalta.

5.3 Toteutus

5.3.1 Lähtökohdat

Käyttöliittymä toteutetaan CODESYSin antamien työkalujen puitteissa. Käyttöliittymä koostuu laajalti kasasta Visualization-sivuja. Yksi Visualization-sivu sisältää graafisia elementtejä ja eri toimintoja. Lisäksi sille voidaan määritellä omia muuttuja, jotka esitellään samalla tavalla kuin tavallisissa POU:ssa. Visualization-sivujen lisäksi käyttöliittymäpuolella on myös omia POU:sta koostuvia koodilohkoja sekä muuttujalistoja, jotka tukevat käyttöliittymän toimintaa.

Visualization Managerista säädetään käyttöliittymän tiettyjä ominaisuuksia, tärkeimpänä se, miltä sivulta käyttöliittymä aloittaa, kun se käynnistetään. Lisäksi sieltä voidaan määrittää, mitkä sivuista ylipäätänsä ladataan ajoon, sekä paljonko muistia käyttöliittymälle annetaan.

CODESYS on aikaisemmin käyttäytynyt oudosti muistin kanssa tekijän edellisessä projektissa: ohjelma antaa outoa virheilmoitusta ajon yhteydessä ja ainoastaan muistin lisääminen on auttanut asiaan. Eli muistia lisätään tarvittaessa, mutta muuten siihen ei tarvitse koskea.

Koska muisti on aiheuttanut pieniä ongelmia, lähdetään siitä näkökulmasta, että yritetään saada tuotoksesta kevyempi: eli vähemmän Visualization-sivuja ja niiden elementtimäärää pienemmäksi mahdollisuuksien mukaan.

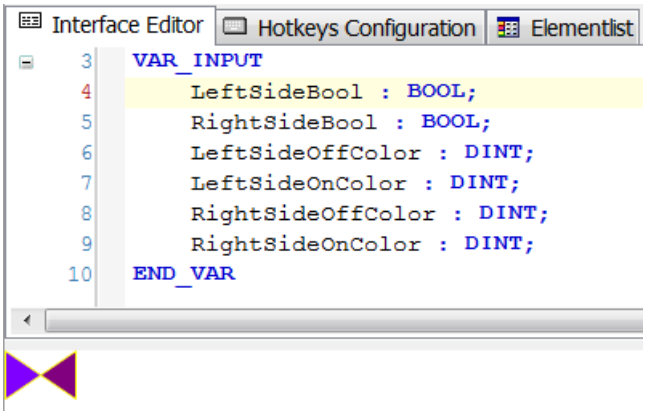
5.3.2 Visualisaation elementtityypit

Käyttöliittymän sivut koostuvat eri määrästä erilaisia elementtejä. Näitä elementtejä ovat esimerkiksi yksinkertaiset graafiset kuviot kuten neliöt, polygonit ja viivat. Näiden lisäksi on tarjolla tekstielementtejä kuten otsikkoelementti ja monipuolisempi tekstikenttä.

Yksi tärkeimmistä elementtityypeistä on kehys, joka voi kutsua muita elementtejä itseensä. Tämä on avain käyttöliittymän liiallisen raskauden välttämiseksi. Kuvaelementti on myös kätevä, mutta tässä projektissa niitä ei ole. Sen sijaan kaikki ”kuvat” kuten suodattimen grafiikka, tehdään edellä mainittujen polygonien ja viivojen avulla.

5.3.3 Alielementit

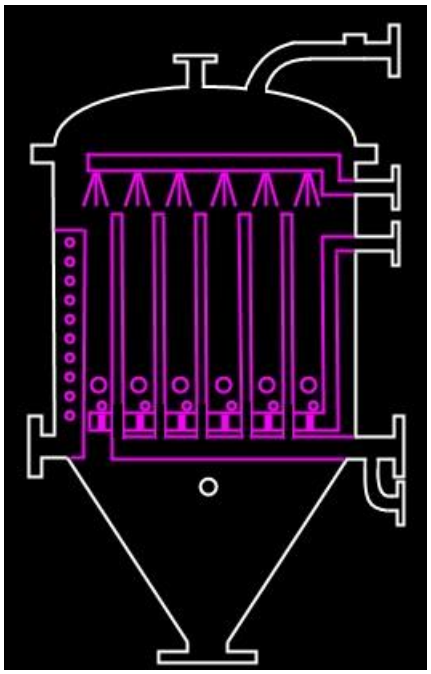
Alielementit ovat elementtejä, joita edellä mainitut kehykset kutsuvat. Se on oma Visualization-sivunsa, jolle rakennetaan samoja asioita ja ominaisuuksia kuin varsinaisille sivuille. Alielementin idea on, että se tarvitsee suunnitella vain kerran, ja sitten sitä voidaan käyttää niin monta kertaa kuin tarvitaan. Tämä on hyödyllistä, koska jotkut alielementit sisältävät niin paljon elementtejä, jolloin niiden tekeminen uudestaan ja uudestaan olisi hyvin kankeata, ja ohjelman koko paisuisi valtavasti.



Kuva 10. Venttiilin grafiikka, sekä sen esitetyt muuttujat.

Esimerkkinä alielementistä kuvassa 10 on vaakatasossa oleva venttiili. Se on muodostettu kahdesta kolmion muotoisesta polygonista. Molemmille polygoneille on annettu ja määritelty boolean-muuttuja sekä kaksi double-muuttujaa väreille. Double-muuttujiin määritellään esimerkiksi heksadesimaaleilla värikoodit. Kuvassa venttiilin värit ovat violetin eri sävyjä. Jos ne näkyvät ajossa, niin silloin niille ei ole annettu muuttujaa tai jotain on mennyt pahasti pieleen.

Kun alielementti on valmis, pääsivuilla kehyselementti voi kutsua sellaisen. Kehys näkee alielementin esitellyt muuttujat referensseinä, joihin voidaan sitten kytkeä halutut arvot tai muuttujat. Venttiilin tapauksessa väreiksi annetaan punainen ja vihreä (joka indikoi onko suljettu vai avattu), sekä boolean-muuttujiksi ohjauspuolelta valitun venttiilin muuttuja, joka kertoo, onko se päällä.

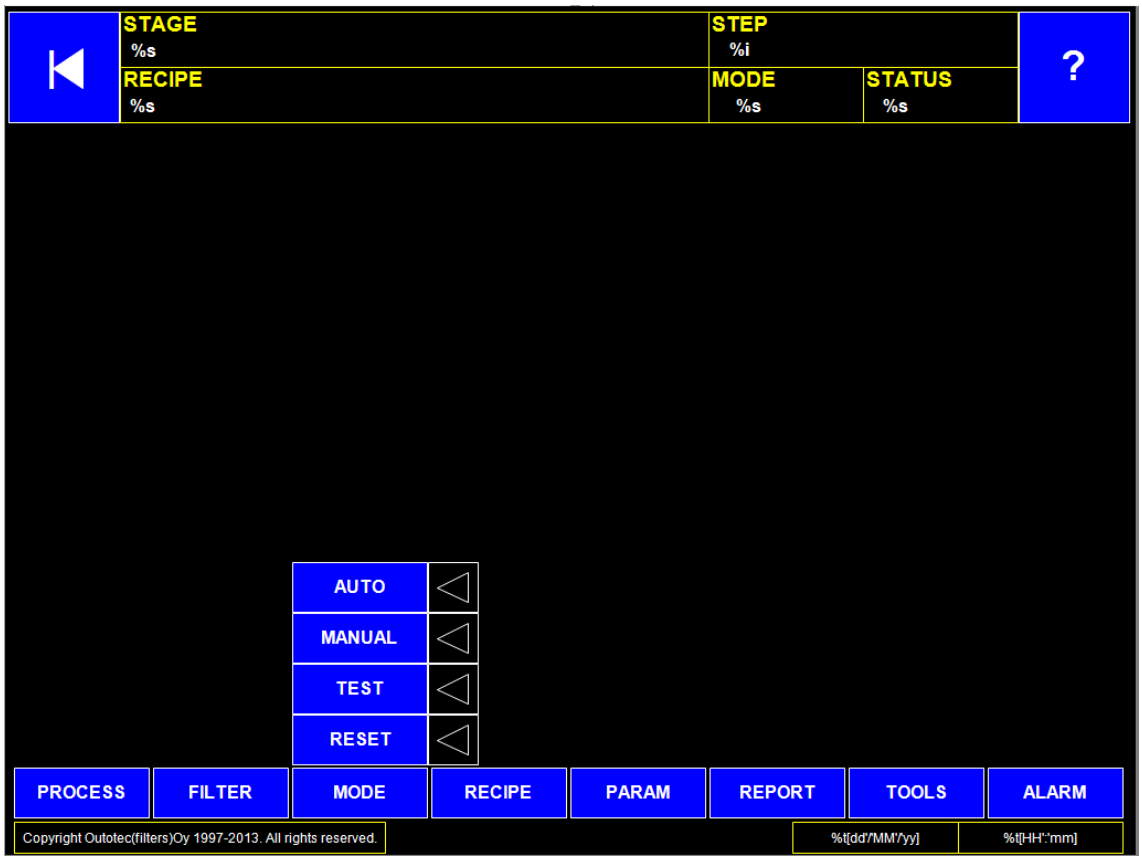


Kuva 11. Käyttöliittymäeditorin elementeillä rakennettu ”kuva”.

Erikoismaininnan saa alielementeistä uutta käyttöliittymää varten tehdyt ”kuvat”. Näitä jouduttiin tekemään pari kappaletta, koska vanhan Proface-version kuvat olivat liian pieniä ja huonoresoluutioisia. Näihin vierähti aikaa kokonainen työpäivä, mutta lopputulos näyttää hyvältä ja sopii hyvin yhteen muun ulkoasun kanssa.

5.3.4 Navigointikehykset

Varsinaisia Visualization-sivuja lähtiessäni suunnittelemaan perehdyin ensin vanhan Proface-version ulkoasuun. Tutkin, mitä sivuja siinä on ja mitä yhteistä niillä on. Sivujen teemat on käsitelty kappaleessa 5.2. Sivujen navigointipalkit ovat pitkälti samannäköisiä, joista on pari variaatiota. Loogisinta on siis tehdä niistä alielementtejä, joita kutsutaan kehyselementeillä.



Kuva 12. Pääkehys

Kuvassa näkyy pääkehys, joka kattaa hyvin kaikki tärkeimmät navigointinapit. Alapalkissa näkyvät napit eri pääsivuille. Painamalla sivu vaihtuu. Mode-nappi sulkee ja avaa kuvassa näkyvät valintanapit. Valitun moodin indikaattori näkyy vihreänä. Yläpalkissa vasemmassa ylänurkassa on paluunappi, joka palaa edellisille sivuille. Oikeasta yläkulmasta päästään ohjesivulle, jonka sisältö riippuu, miltä sivulle sinne ollaan tultu.

Pohjalla nähdään tekstilaatikot ajannäyttämiseen, joissa näkyy niiden formaatti; ensimmäisessä päivä, kuukausi ja vuosi, ja jälkimmäisessä tunnit ja minuutit.

Yläpalkeissa näkyy suodattimen sen hetkisen tilan mittareita. Stage eli vaihe kertoo, missä vaiheessa suodatin on. Step eli askel kertoo, missä vaiheen askeleessa mennään. Recipe kertoo, mikä resepti on valittu. Mode kertoo, mikä moodi valittu, ja Status kertoo, onko suodatin ajossa, valmis ajoon, resetoitu tai pysäytetty.

Kuvan 12 kehys on kaikista kehyksistä monimutkaisin, reseptisivulla on pieni variaatio, jossa reseptin tekstilaatikossa näytetään valittavaa reseptiä, eikä sitä, mikä on sillä hetkellä ajossa. Ohjesivuille on myös omat kehyksensä. Näissä alanapit on poistettu kokonaan, jäljelle on jätetty vain ajannäyttö, sekä yläpalkkiin paluunappi ja linkki riippuen paikasta. Kaiken kaikkiaan navigointikehyksiä muodostui kuusi kappaletta.

5.3.5 Ohjesivut

Käyttäjälle avuksi on kirjoitettu ohjesivut. Jokaiselle pääsivulle on oma ohjesivu, joka kertoo lyhyesti, mitä sivun funktiot ja lyhenteet ovat. Pääsivujen ohjesivu on kuitenkin toteutettu niin, että on oikeasti vain yksi sivu. Sivun sisältö riippuu mistä linkistä on sivulle menty, jolloin haluttu oikea teksti näkyy. Ohjetekstit on tallennettu yhteen tekstilistaan. Tekstilista koostuu indeksistä ja sisällöstä.

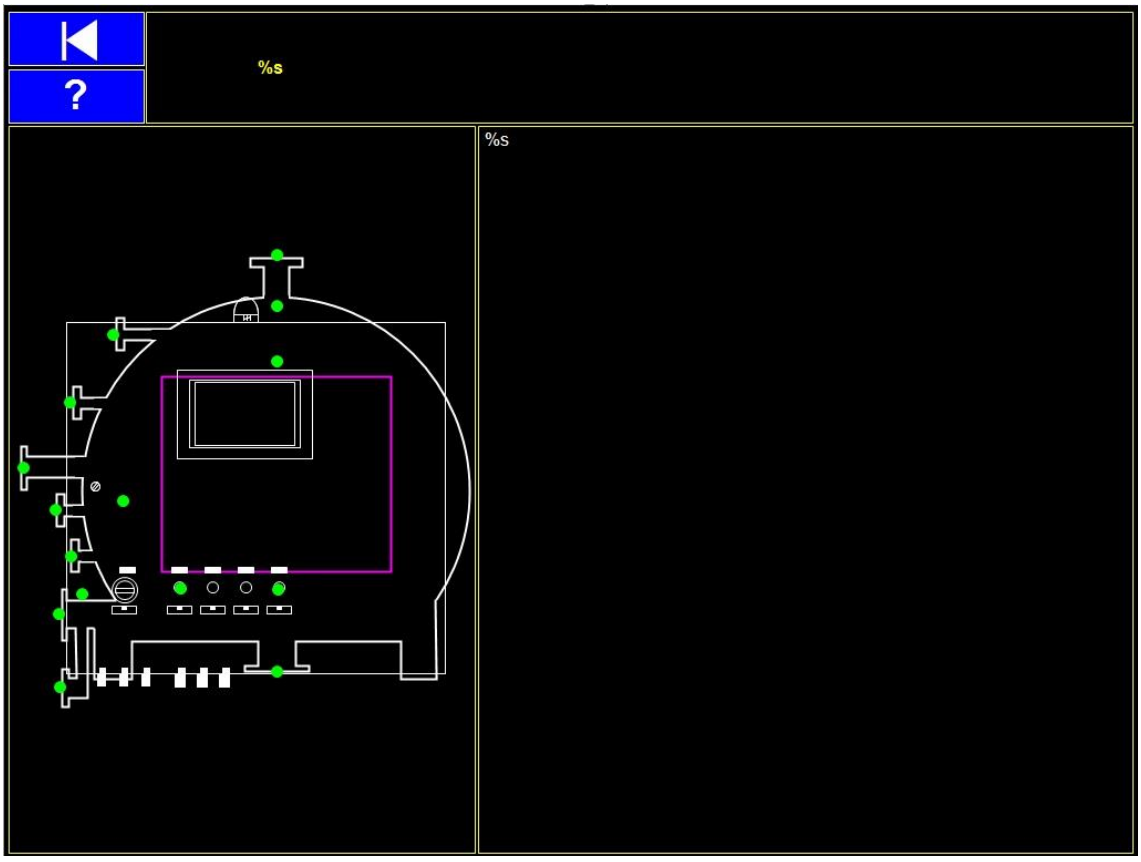
ID	Default
0	PROCESS screen displays the main devices ...
1	FILTER screen displays the polishing filter wi...
2	RECIPE screen displays the filtering recipes ...
3	PARAMETER screen displays the filter contr...
4	CYCLE REPORT screen displays the report ...
5	CHOOSE LANGUAGE is used to choose the...
6	Once the system discovers a new alarm, an ...
7	ALARM HISTORY screen displays the 15 lat...
8	When TEST mode is selected, the TEST M...
9	CONFIGURATION screen is used to choose ...
10	PROCESS PRESSURE REPORT screen dis...

Kuva 13. Pääsivujen ohjetekstit indekseineen

Eli jos käyttäjä haluaa lukea filtterisivusta, painaa hän oikeaa linkkiä joko pääsivun oikeasta yläreunasta tai ohjesivujen valikosta, jolloin asianmukainen muuttuja muuttaa arvokseen 1, ja ohjesivun tekstikenttä näyttää sen indeksin sisällön.

Edellisten lisäksi on pieni manuaali, jolla on neljä ”sivua”. Ne ovat samaan tapaan tehty kuin kuvassa 8, sijoitettuna omassa tekstilistassaan. Toisin kuin pääsivujen ohjesivussa, manuaalisivulla on kaikki neljä tekstiä samaan aikaan. Erona on, että vain yksi näkyy kerrallaan, ja muut ovat piilotettuna. Manuaalisivulla on oma eteen- ja taaksepäin-nappi, joka säätelee Visu_MenuPage- muuttujaa. Tämä määrittää, mikä teksteistä näkyy sillä hetkellä.

Viimeinen kokonaisuus on eri hälytysten ohjesivut. Eri hälytyksiä on 38 kappaletta, ja jokaiselle on jälleen kuvan 8 mukaisesti tekstilista, jossa on indeksi ja teksti jokaiselle hälytykselle. Hälytykset on jaettu kahdelle sisällyssivulle, joissa on linkit hälytysten ohjesivuille sekä indikaattorivalot, jotka palavat punaisena, jos kyseinen hälytys on aktiivinen.



Kuva 14. Hälytysten ohjesivu editointitilassa

Kuvassa 9 on hälytysten ohjesivun toteutus. Sivun tekeminen jokaiselle eri hälytykselle tuottaisi aivan liikaa kuormaa, joten toteutus on tehty yhden sivun varaan. Kuvassa vasemmalla näkyy kolme eri alielementtiä, suodatin ja kaksi paneelia. Näistä näkyy yksi

tai ei mikään, riippuen mitä hälytystä katsotaan. Teksti ja otsikko tulevat kahdesta tekstilistasta, joiden indeksit täsmäävät niin, että yhdellä hälytyksellä on sama numero. Vihreät ympyrät kuvaavat, missä päin suodatinta vika esiintyy, ja vain yksi niistä näkyy per hälytys. Se myös vilkkuu taustalle rakennetun logiikan ansiosta. Sivun toteutus antaa kevyen lopputuloksen, mutta myöhempien muutosten teko on työlästä, kun kaikki on ahdettu yhteen editoitavaan ruutuun.

5.4 Testaus ja lopullinen kokoonpano

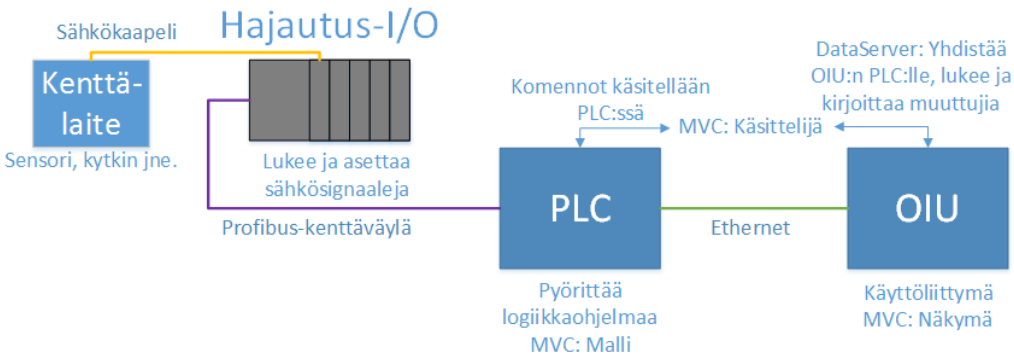
Pelkän käyttöliittymän testaaminen on varsin suoraviivainen tehtävä. Käyttöliittymä ladataan työkoneella pyörivään CODESYS Control Win V3 – Soft-PLC:hen, joka sitten avaa ikkunan, jossa käyttöliittymä pyörii ja sitä voidaan käyttää. Se toimii ilman erityisiä asetuksia, jolloin testaaminen helpottuu.

Testaus paikallisesti on yksinkertaista, mutta miltä lopullinen kokoonpano oikeilla laitteilla näyttää? CODESYS-projekti on pilkottu kahteen applikaatioon eli ohjelmaan, jotka ovat PLC-logiikka ja OIU-käyttöliittymä. Ohjelmat ladataan kahteen erilliseen tietokoneeseen, jotka ovat Janztecin emPC-CX+-tyyppinen teollisuus-PC sekä myös Janztecin emVIEW-15T-mallinen paneeli-PC. Molemmissa koneissa pyörii 32-bittinen Ubuntu Linux -käyttöjärjestelmä. Koneisiin ei ole asennettu kuin (Ubuntun oletusohjelmien lisäksi) CODESYSin runtime -ohjelmisto, joka pyörittää ohjelmia, sekä joitain käynnistyksen yhteydessä ajettavia skriptejä, pari ajuria sekä ulkonäöllisiä muutoksia.

Kuva 15. Janztec emPC-CX+ vasemmalla ja oikealla emVIEW-15T

Eli vain PLC-logiikka ladataan teollisuus-PC:hen, ja vain pelkästään OIU-käyttöliittymä paneeli-PC:hen. CODESYS-runtime pyörittää ohjelmissa määriteltyjä Taskeja, joita voidaan verrata perinteisen tietokoneohjelman Main-ohjelmaan. PLC-ohjelmalla on vain yksi Task, joka pyörittää koko logiikkaa. OIU:lla taasen on enemmän Taskeja. Ensimmäinen on PLC:n tapainen, joka pyörittää joitain itse ohjelmoituja käyttöliittymän yksityiskohtia. Loput ovatkin automaattisesti luotuja Taskeja, joita ovat: hälytysten manageri, DataServerin pyörittäjä sekä visualisaation hallinnoija. Taskien konfiguraatiossa on käytännössä vain pari asetusta: mitä ohjelmalohkoa kutsutaan, Taskin prioriteetti ja sykliväli. Sykliväli on näistä oleellisin ja se vaihtelee Taskeissa.

PLC:ssä logiikka käydään läpi 20 millisekunnin välein, eli yhden sekunnin aikana koko ohjelma käydään läpi peräti 50 kertaa. OIU:n Taskit suoriutuvat harvemmilla sykleillä. Hälytykset käydään läpi 50 ms välein, DataServer päivittyy 200 ms välein eli se noutaa ja lähettää PLC:lle muuttujat 5 kertaa sekunnissa. Visualisaatio päivitetään 100 ms:n välein, eli ruutu päivittyy 10 kertaa sekunnissa. Hitaammat syklit näillä johtuvat siitä, etteivät ne ole niin kriittisiä nopeuden kannalta. Lisäksi paneeli-PC:ssä ei ole niin hyvä suoritin verrattuna emPC-CX+:aan (jossa on i3-suoritin).



Kuva 16. Järjestelmän tiedonsiirron kaavio, ohjelmat ja MVC:n osat

Koneet ovat yhdistetty samaan verkkoon Ethernetin välityksellä. Paneelin ohjelman DataServer, joka hallinnoi ohjelmien välistä kommunikaatiota, osaa ottaa automaattisesti yhteyden logiikkaohjelmaan ja välittää sen jälkeen muuttujien arvoja edestakaisin. Yhteys hajautus-IO:ihin tapahtuu logiikka-PC:n kautta Profibus-kenttäväylän kautta. Koneessa on yksi Profibus-kortti, joka suorittaa tämän tehtävän. Hajautus-IO:t eivät pyöritä mitään ohjelmaa sisällään, vaan ne yksinkertaisesti ylläpitävät digitaali- ja analogiarvoja niistä kenttälaitteista, joihin ne on kytketty. Tämä on merkittävin ero, mikä

erottaa hajautus-IO:t tavallisista PLC-koneista, joissa on suoritin, ja ne ajavat logiikkaa, kuten esimerkiksi Siemensin tai Allen-Bradleyn mallit.

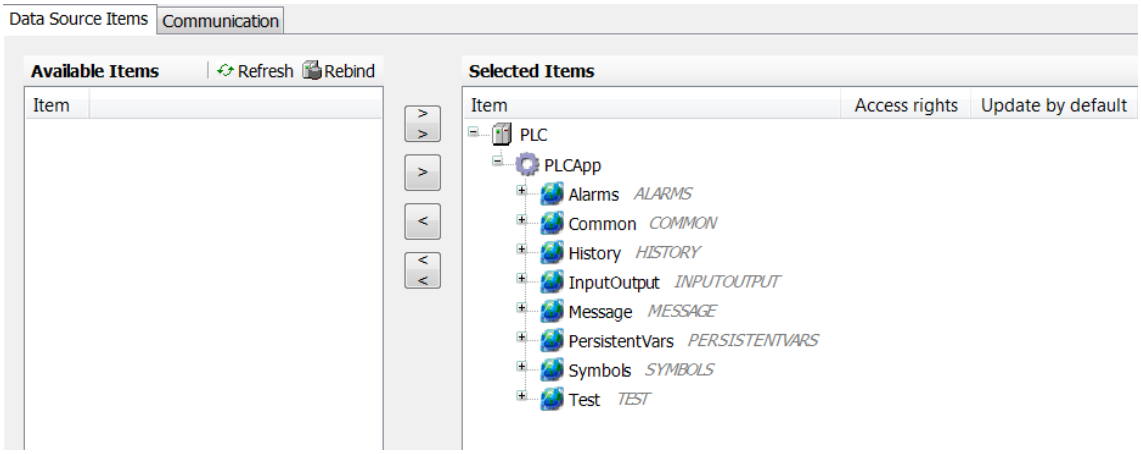
Palaten MVC-arkkitehtuuriin oheisessa kaaviossa on kuvattu, miten se toteutuu projektissa. Mallia pyöritetään logiikka-PC:ssä ja näkymää paneeli-PC:ssä. Käsittelijän toteutuminen on hankalampi asia. Esimerkiksi Auto-moodi-nappia painamalla siihen kytketty muuttuja asetetaan TRUE:ksi. DataServer paneeli-PC:ssä välittää tämän logiikalle Ethernet-yhteyden kautta, jossa itse malli käsittelee toiminnon. Eli tältä osin MVC-malli ei toteudu mielestäni täydellisesti, koska käsittelijän osuus jää turhankin suppeaksi.

Testailua suoritetaan ensin testaamalla navigointi: Jokaista nappia, jolla voidaan siirtyä toiselle sivulle, painetaan, jotta nähdään, että ne toimivat ja vievät oikealle sivulle. Elementit, joilla on ON/OFF-tila, kuten esimerkiksi venttiilit ja indikaattorivalot, kytketään käyttöliittymässä esitellyllä boolean-tyyppisellä ”Dummy”-muuttujalla. Muuttujan tilaa muutetaan sitten ohjelmistoympäristön debug-tilassa, ja katsotaan, miten elementit käyttäytyvät ajossa.

Myös numeeriset elementit pitää testata, ja se tehdään samalla tavalla kuin boolean-tyyppisetkin. Numeroelementteihin kytketään Int- tai Real-tyyppinen muuttuja, jolle annetaan arvoja. Näkymää sitten tutkitaan ja katsotaan, että elementit näyttävät annetun arvon oikein. Lisäksi elementit eivät saa venyä sallitun alueen ulkopuolelle, eli varmistetaan, että ne on määritelty oikein.

5.5 Ohjelman ja käyttöliittymän kytkentä

Kun ohjelma ja käyttöliittymä olivat kutakuinkin kasassa (pois lukien hälytykset), oli aika yhdistää ne. Yhdistäminen toteutetaan luomalla käyttöliittymän puolelle DataServer eli datapalvelin. Nimi on hieman harhaanjohtava koska kyseessä ei ole mikään erillinen palvelin, vaan se on CODESYSin sisäinen konfiguraatiotyökalu. Tällä valitaan, mitä muuttujatietoja siirretään logiikka- ja käyttöliittymäkoneen välillä, sekä asiaan liittyvät konfiguraatiot kommunikaation osalta kuten osoitteet. DataServer konfiguroidaan käyttöliittymän puolelle.



Kuva 17. DataServerin näkymä, oikealla näkyy valitut muuttujalistat muuttujineen, jotka välitetään käyttöliittymälle

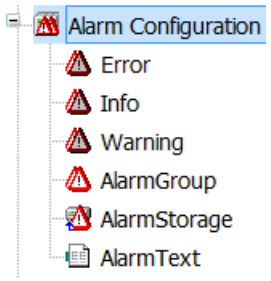
Kuva havainnollistaa DataServerin sisältöä, vasen puoli näyttää tyhjää, toiminnassa siinä on ohjelmapuolen kaikki muuttujat, joista poimitaan ne, jotka halutaan välittää käyttöliittymälle. Luonnollisesti käyttöliittymä voi välittää näiden muuttujien tiloja takaisinpäin ohjelmalle, kuten esimerkiksi reseptin vaihdon.

Lopuksi käyttöliittymän elementteihin vaihdetaan oikeat muuttujat testausmuuttujien tilalle. Ohjelmapuolelle rakennettiin simulaatiokoodi, joka mukailee oikean laitteen käyttäytymistä, ja kokonaisuutta pystyttiin testaamaan käytännössä.

6 Hälytykset

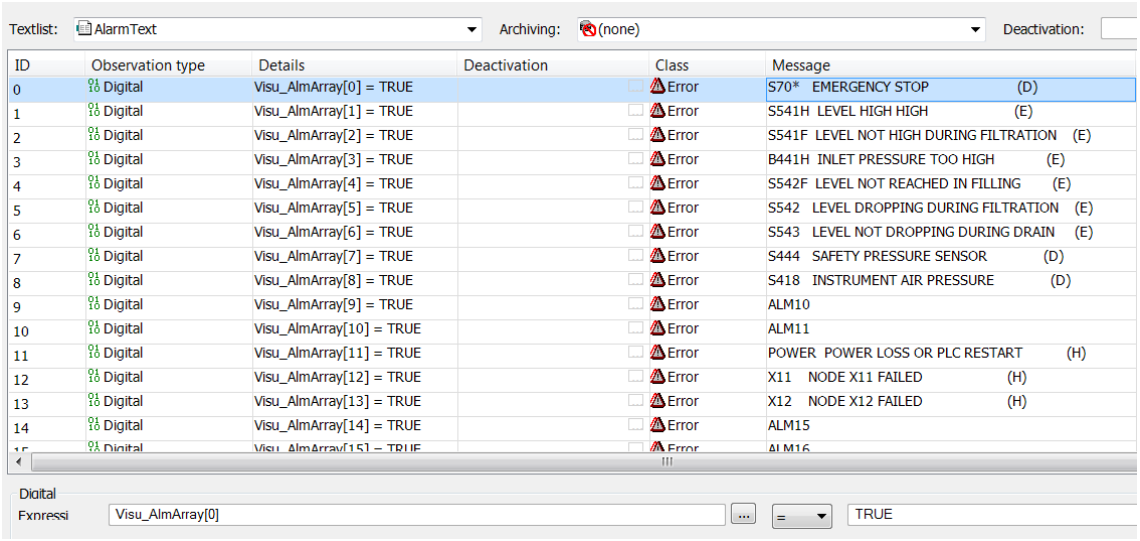
6.1 Taustakoodi

Työn viimeinen osuus on tehdä hälytysjärjestelmä, joka ilmoittaa käyttäjälle mahdollisista hälytyksistä. Toteutus rakennetaan käyttöliittymäpuolelle, jonne luodaan uusi Alarm Configuration.



Kuva 18. Alarm Configuration, hälytysten osat

Kuvasta nähdään, mitä osia hälytyksille tehdään. Oleelliset osat ovat AlarmGroup ja AlarmText-tekstilista. Error-, Info- ja Warning-sivuilta voidaan säätää näiden ominaisuuksia tarkemmin, mutta tähän työhön niitä ei tarvitse säädellä.



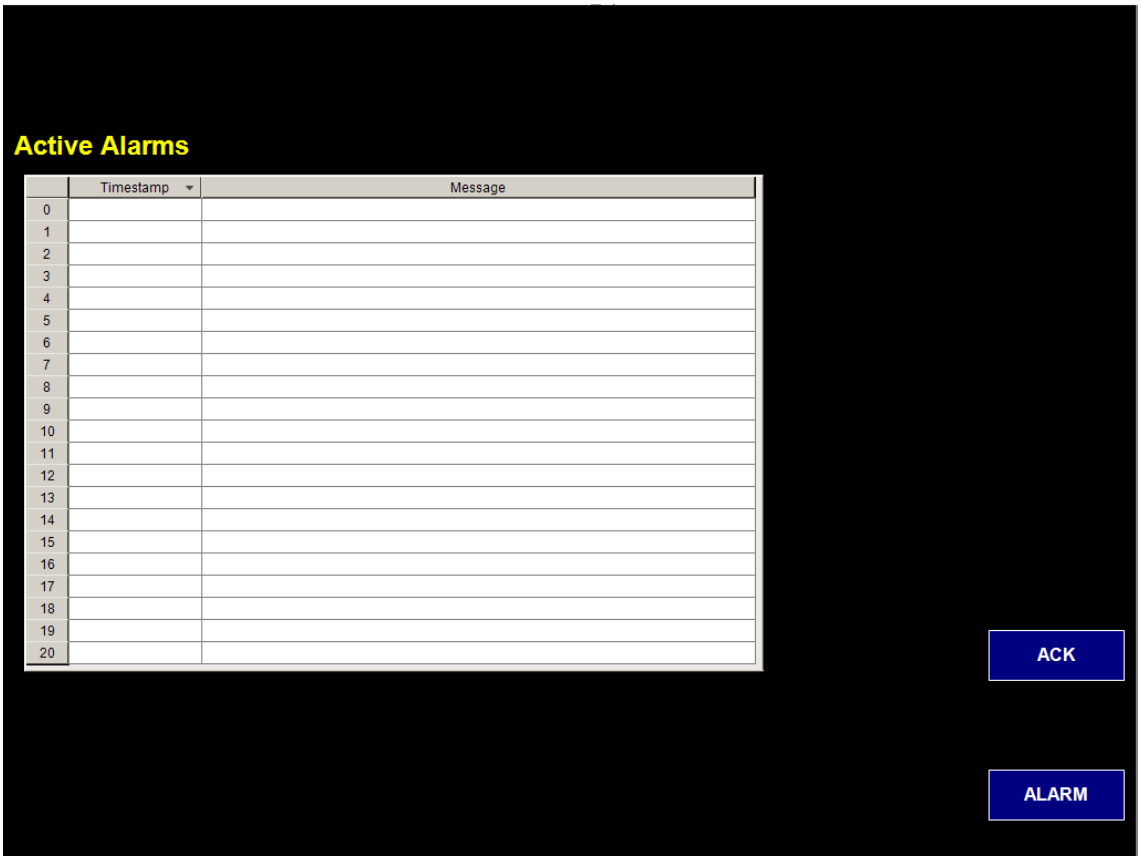
Kuva 19. AlarmGroupin sisältöä

AlarmGrouppiin määritellään, mistä muuttujista halutaan saada hälytyksiä. Kaikki hälytykset tuodaan tässä tapauksessa Visu_AlmArray-taulukosta, joka on taas kytketty ohjelmalogiikan Alarms.Alm-muuttujaan, josta on kerrottu luvussa 4.3.1.

Kaikki hälytykset on määritelty Error-tyyppiseksi, ja niiden kuvaukset tuodaan AlarmText-tekstilistasta, mikä näkyy valittuna kuvassa. Viimeinen huomioitava seikka on hälytysten tyyppi, joka on kaikille valittu digitaaliseksi. Tyyppejä on erilaisia, kuten ylä- ja alaraja eli hälytys voisi laueta jos arvo menee tietyn rajan yli. Mutta tässä työssä kaikki ovat digitaalisia, eli ne ovat yksinkertaisia ON/OFF-hälytyksiä. Jos jonkin hälytyksen bitti muuttuu TRUE:ksi, hälytys laukeaa.

6.2 Käyttöliittymäpuoli

Alarm Configurationin teko nähdään käytännössä käyttöliittymässä, jossa on tehty hälytyksille oma sivu. Valitettavasti tämä ominaisuus CODESYSissä ei toimi kovin hyvin. Vanhassa Proface-käyttöliittymässä hälytyslista on hyvin muotoiltu, sekä on myös erillinen hälytyshistoriasivu. Uudessa versiossa hälytyslistaa, joka näyttää kovasti tavalliselta Excel-taulukolta, ei voida mitenkään ulkonäöllisesti muuttaa. Lopputulos näyttää kovin huonolta.



Kuva 20. Hälytyssivu käyttöliittymässä, ilman navigointikehystä

Murheet eivät lopu siihen. CODESYSillä ei voi tehdä erillistä hälytysten historialistaa, joten sitä sivua ei tehdä ollenkaan. Tavallisessa hälytyslistassa on kyllä ominaisuus, joka näyttää historian, mutta valitettavasti se ei toimi ollenkaan. Vielä pahempi on kuittausjärjestelmä. Kun hälytys aktivoituu, se ilmestyy taulukkoon punaisella. Se voidaan silloin valita ja kuvan ACK – eli kuittausnappia painamalla hälytys raukeaa. Tämäkään ei toimi kunnolla, joka varmasti tulee aiheuttamaan harmia myöhemmin laitetestauksessa.

6.3 Banneri

Viimeinen ominaisuus, koskien viestejä,on banneri. Tämä on Proface-käyttöliittymän ominaisuus, eikä siitä ole CODESYSissä vastaavaa versiota. Se pitää siis rakentaa itse. Bannerillä esitetään puhtaiden hälytysten sijaan ”messageja” eli viestejä jotka ovat informatiivisia viestejä, kuten ”Odottaa käynnistyslupaa” ja ”Suodatin manuaalitilassa”. Viestit ovat Profacessa hälytysten kanssa samoissa listoissa, mutta CODESYSissä ne erotellaan erillisiin listoihin.

Banneri on käytännössä punaisella pohjalla oleva laatikko, joka sisältää tekstiä. Se sijoitetaan navigointikehyksiin, jolloin banneri näkyy varmasti joka sivulla. Bannerin ulkonäköön ei sen ihmeemmin kiinnitetä huomiota korkeintaan fonttia säädetään. Oleellinen toimivuus pitää rakentaa taustalle käyttöliittymän koodiin.

Ohjelmapuolen koodiin rakennetaan aliohjelma, joka skannaa hälytykset joka syklillä for-luupissa. Jos luupissa havaitaan uusi hälytys, asetetaan AlmNew-boolean-muuttuja todeksi. Tällöin käyttöliittymä saa tiedon tämän muuttujan tilasta ja banneri muuttuu näkyväksi. Se säilyy näkyvänä niin kauan, kunnes käyttäjä käy hälytyssivulla kuittaamassa hälytyksen. Bannerille rakennettiin myös oma luuppi käyttöliittymäpuolelle, joka käy äsken mainitun aliohjelman tavoin hälytyslistan läpi.

Valitettavasti bannerin toiminta ajon aikana ei vakuuttanut. Jos vain yksi viesti oli päällä, niin siinä ei ollut ongelmia. Ongelma tulee, jos useampi on päällä, jolloin banneri näyttää ilmoituksia epätasaisin välein. Tarkoitus oli, että yhtä ilmoitusta näytettäisiin kolmisen sekuntia ja siiten siirryttäisiin seuraavaan ilmoitukseen. Tilannetta yritettiin korjata synkronoimalla ohjelma- ja käyttöliittymäpuolen päivityssyklit samaksi, mutta se ei auttanut. Lisäksi banneri vielä aiheutti selvää tökkimistä käyttöliittymässä.

Lopputilanteessa banneri jouduttiin jättämään kokonaan pois, työn deadline oli lähestymässä ja projekti piti saada virheettömänä laitetestaukseen. Hälytysten ilmoitus on tehty yksinkertaisemmin. Jos uusi hälytys laukeaa, käyttöliittymä suunnistaa tällöin itsestään hälytyssivulle. Ratkaisu on hieman karkea, mutta se toimii ja samalla tavalla kuin Profacessa.

7 Yhteenveto ja pohdintaa

Insinöörityön tarkoituksena oli päivittää suodattimen prosessi Siemensin Simatic-ohjelmistoalustalta 3S-Softwaren CODESYS-alustalle sekä Proface-ohjelmistolla tehty käyttöliittymä myös CODESYS-alustalle. Itse olin tehnyt jo aikaisemmin Etteplanille pari projektia, joissa CODESYS:iä käytettiin. Näissä projekteissa vastasin käyttöliittymien suunnittelusta, joten niiden teko oli jo tuttua hommaa tämän projektin alkaessa. Ohjelmapuolen tekeminen vaati IEC 61131 -standardin sekä CODESYS:in toimintojen opiskelua. Lisäksi piti perehtyä Simatic-ohjelmistoon ja miten siinä asiat toimivat verrattuna CODESYS:iin.

Työssä lähdettiin liikkeelle aluksi käyttöliittymästä, jota lähdettiin rakentamaan vanha versio mallina. Käyttöliittymä rakennettiin CODESYS:in omalla käyttöliittymäeditorilla, jolla saatiin tehtyä hyvin vastaavan näköinen lopputulos verrattuna alkuperäiseen. Seuraavaksi luotiin ohjelmalogiikka, joka rakennettiin vanhan ohjelman määritysten mukaisesti. Osa funktioista, kuten aikafunktiot rakennettiin uusiksi, CODESYS:in omilla toteutuksilla. Lopuksi projektia valvova insinööri, joka tunsi tarkasti suodattimen toiminnan ennestään, tarkasti ohjelmalogiikan.

Projektille asetetut tavoitteet saavutettiin mielestäni riittävän hyvin. Uusi käyttöliittymä muistuttaa hyvin vanhaa versiota, mutta tarkemmalta. Lopputulos on tarpeeksi sulava, omien testausten perusteella. Pienet puutteet, kuten bannerin puuttuminen tietysti harmittaa, mutta kokonaisuuden kannalta se ei haittaa. Hälytysten kuittausongelma oli hieman isompi ongelma, mutta siihen ei ehditty perehtyä kovin hyvin enää, kun ohjelmat piti jo saada testaukseen laitehallille, missä varsinainen suodatin kootaan ja testataan.

Tätä kirjoittaessani suodatin on jo toimitettu loppuasiakkaalle, ja kuulemani mukaan kaikki toimii hyvin ohjelman osalta. Edellä mainittu hälytysongelmakin, jossa hälytystä ei voitu kuitata siihen tarkoitetusta napista, oli saatu korjattua ohjelmistopäivityksen yhteydessä. Banneri korvattiin lopulta pienellä automaattisesti ilmestyvällä popup-ikkunalla, joka näyttää kaikki viestit, ja aktiiviset viestit ovat valaistuja. Ratkaisu toimi tässä tapauksessa, kun viestejä ei ole kuin viisi kappaletta mutta muissa suodatintyypeissä viestejä on huomattavasti enemmän. Tätä projektia ajatellen jatkokehitysmahdollisuudet ovat mielestäni rajalliset. Oikeastaan isoimmat kehitysmahdollisuudet koskevat lähinnä ohjelmistopäivityksiä, joilla korjattaisiin

mahdollisia bugeja pois. Toisaalta en näe, miksei tätä versiota voisi toimittaa muille asiakkaille.

Projekti tehtiin, koska Siemensin version toimittaminen loppuasiakkaalle ei ollut mahdollista. Kysymys kuuluukin, onko CODESYS varteenotettava vaihtoehto Siemensille? Ainakin kun projektia tein, sanoisin ei, koska alustaa riivasivat silloin pienet bugit. Tätä kirjoittaessani alustalle kuitenkin on tullut jo monta päivitystä (uusin versio 3.5 SP8, projektissa 3.5 SP3 Patch 1), joten nähtäväksi jää, onko pahimmista ongelmista päästy eroon. Asian ratkaisee lopulta se, saadaanko lopputuotteesta tarpeeksi luotettava käytössä. Voi olla, että nykyisistä tuotteista viedään Siemensillä tehtyjä versiota, kun ne on jo valmiiksi tehty. Yrityksessä kuulemani huhujen mukaan CODESYSiä saatettaisiin käyttää jatkossa enemmänkin. Yksi merkittävä ero alustoilla on niiden hinta. Alan suuri toimija Siemens toimittaa ohjelmistot ja laitteet kaikki itse, ja lisenssimaksut ovat huomattavat. CODESYSin lisenssimaksut ovat taas lähes mitättömiä, ja erikseen ostettavat laitteet tulevat halvemmaksi verrattuna Siemensin omiin laitteisiin. Eli jos CODESYSin luotettavuus nousee tai on jo riittävällä tasolla, voitaisiin sillä luoda uusia prosesseja uusille tuotteille.

Lähteet

1 John K-H. & Tiegelkamp M. 2010. IEC 61131-3: Programming Industrial Automation Systems

2 Codesys Facts and figures. 3S Smart Software Solutions GmbH. Verkkodokumentti<http://www.codesys.com/company/about-us-facts-and-figures.html>. Luettu 11.5.2016

3 Modular WAGO-I/O-SYSTEM, IP 20 (750/753 Series). Verkkodokumentti. Wago.<http://global.wago.com/en/products/product-catalog/components-automation/modular-io-system-series-750-753/overview/index.jsp>. Luettu 11.5.2016

4 Outotec Larox LSF. Verkkodokumentti. Outotec Oyj. <http://www.outotec.com/en/Products--services/Process-equipment/Filters/Polishing-filters/Outotec-Larox-LSF-/>. Luettu 11.5.2016

5 Outotec Larox LSF. PDF. Outotec Oyj. <http://www.outotec.com/ImageVault-Files/id_762/d_1/cf_2/OTE_Outotec_Larox_LSF_eng_web.PDF>. Luettu 11.5.2016

6 emPC-CX+. Verkkodokumentti. Janztec.<https://www.janztec.com/en/products/embedded-computing/empc/empc-cx/>. Luettu 11.5.2016

7 emVIEW-15T. Verkkodokumentti. Janztec.<https://www.janztec.com/en/products/embedded-computing/emview/emview-15t/>. Luettu 11.5.2016

8 Myers, G J., Badgett, T., Thomas, T M. & Sandler C. 2011. The art of software testing. s. 16.

9 Buschmann F., Meunier R., Rohnert H., Sommerlad P. & Stal M. 1996. Pattern-Oriented Software Architecture Volume 1: A System of Patterns Volume 1 Edition. s. 125-143.